GEATbx.com

Genetic and Evolutionary Algorithm Toolbox for Matlab

GEATbx: Options

Parameter Options

by:
Hartmut Pohlheim

GEATbx version 3.8
(December 2006)

www.geatbx.com
support@geatbx.com

Contents

1 Introduction

The Genetic and Evolutionary Algorithm Toolbox for use with Matlab (GEATbx) provides a framework to utilize Evolutionary Algorithms for the optimization of a broad range of systems. Multiple Evolutionary Algorithms, evolutionary operators and special principles (population models, competition, multiple strategies, visualization) are available.

The GEATbx integrates all these operators and extensions. The user selects preconfigured algorithms or defines its own parameters. All operators and options of the GEATbx can be accessed by setting the appropriate parameters in the Option Structure. The toolbox handles all interactions and dependencies between operators and parameters. It integrates the provided extensions (i.e. visualization, result reports) into a framework for the optimization of real world systems.

The following sections provide an overview of the parameter/option handling of the GEATbx.

Section 1.1 (Predefined Evolutionary Algorithms) describes the preconfigured Evolutionary Algorithms for different system domains called ToolBoX functions. Each of these predefined algorithms is a collection of parameter options. The sum of these options define a special Evolutionary Algorithm. By selecting/using one or multiple of these ToolBoX functions complete algorithms may be applied in one step.

The default option settings of the GEATbx are preconfigured for the parameter optimization of real valued variables. Without selecting any special options or ToolBoX functions the GEATbx works good for this problem domain.

All option handling/applying of the GEATbx is done inside/using one function - `geaoptset`. The handling and working of this function is similar to `simset` from Simulink. How this functions works and special features useful for real work are explained in Section 1.2 (Option handling with geaoptset.m)

Section 1.3 (Examples of option settings) explains, how to define options and their parameters. A few examples of realistic setups apart from the predefined ToolBoX functions in Section 1.1 are provided. Special attention is paid to the definition of multiple strategies for the subpopulations.

During an optimization the GEATbx can output status information. Section 1.4 (Status and result output during optimization) explains this information. Additionally, the options to adjust the frequency and amount of status information are named. The same information may be saved into text files. This is very important for long running optimizations or the analysis of multiple runs. At the same time this produces a good documentation of the optimizations done.

Chapter 2-12 explains all available options of the GEATbx version 3.x in detail. Each chapter describes the options of a group (General, Recombination, Mutation, ...). Only a few of these options are used at a time. Most of these options are only needed for very special cases. The GEATbx was and is used for many different tasks - and some of the options are only needed then. The main objective of defining all these options is, that the toolbox may be completely controlled from outside by the definition of the respective parameters/parameterfiles. This is especially important when using the compiled version of the GEATbx.

The described concept for the option and parameter definition of the GEATbx version 3.x is different from GEATbx version 1.9x and 2.x. To provide an easy upgrade path the explanation of each option contains information of the option number under version 1.9x or 2.x (at the end of each option explanation in Chapter 2-12).

Chapter 13 (Comparison of Options (1.9x / 2.x / 3.x)) contains an tabular overview of the numbers of the available options in GEATbx version 1.9x and 2.x and their names in GEATbx version 3.x (the new names of the options are linked to the respective explanations). All options available in version 1.9x and 2.x are also available in version 3.x. Thus, the upgrade from an older version to the new one involves "just" the renaming of the options and the adaptation of the naming mechanism explained in Section 1.3 (Examples of option settings).

The GEATbx is fully modularized. The operators and functions of the GEATbx may be used for building own Evolutionary Algorithms. All functions use a defined and self-contained interface including check of the validity of the inputs. So, all operators are directly available and can be used inside special functions, algorithms and tools. However, then all extensions for the efficient application to the optimization of real world systems are lost. No direct example for the composition of own algorithms is provided. Nevertheless, the main function of the GEATbx, `geamain2`, is a full blown example.

1.1 Predefined Evolutionary Algorithms

The GEA Toolbox includes a number of predefined Evolutionary Algorithms called ToolBoX functions (***tbx3*.m***). One example is the frequently used `tbx3real`. These functions define some of the options of the GEATbx and define thus a special algorithm. The ToolBoX functions can be used as a starting point for individually predefined Evolutionary Algorithms.

These special option files contain all settings for a given problem class. For instance, optimization of ordering problems (`tbx3perm`) or parameter optimization of integer (`tbx3int`) or binary variables (`tbx3bin`).

`tbx3real` (real valued variables, multiple subpopulations, standard operators for real valued variables):

- real valued variables (`VariableFormat`=0): the EA works on real values as well (genotype==phenotype),
- recombination algorithm: discrete recombination (`Recombination.Name`='`recdis`'),
- mutation algorithm:
  - real valued mutation (`Mutation.Name`='`mutreal`'),
  - mutation range for 6 subpopulations (`Mutation.Range`=[0.1, 0.03, 0.01, 0.003, 0.001, 0.0003]),
  - mutation precision for all subpopulations (`Mutation.Precision`=12).

`tbx3int` (integer valued variables, multiple subpopulation, standard operators for integer valued variables):

- integer valued variables (`VariableFormat`=2): the EA works on integer values as well (genotype == phenotype),
- recombination algorithm: discrete recombination (`Recombination.Name`='`recdis`'),
- mutation algorithm: integer valued mutation (`Mutation.Name`='`mutint`').

`tbx3bin` (binary valued variables, multiple subpopulation, standard operators for binary valued variables):

- binary valued variables (`VariableFormat`=4): the EA works on binary values as well (genotype == phenotype),
- recombination algorithm:
  - double point crossover with reduced surrogate (`Recombination.Name`='`recdprs`'),
  - recombination rate 0.9 (`Recombination.Rate`=0.9),
- mutation algorithm: binary valued mutation (`Mutation.Name`='`mutbin`'),
- In case the problem to solve uses real valued variables (phenotype) and the EA should use binary values (genotype), just set `VariableFormat`=1 and employ the parameters above.

All previous ToolBoX functions use more or less standard parameters. The following examples show more advanced variants.

`tbx3es1` (real valued variables, locally oriented optimization, uses evolution strategy mutation):

- real valued variables (`VariableFormat`=0): for the used mutation algorithm the EA **must** work on real values as well (genotype == phenotype),
- number of individuals: 12 (`NumberIndividuals`=12),
- number of subpopulation: 1 (`NumberSubpopulation`=1),
- selection algorithm:
  - truncation selection (`Selection.Name`='`seltrunc`'),
  - generation gap 1 (`Selection.GenerationGap`=1.0)
  - selection pressure 6 (`Selection.Pressure`=6.0) - selects only the 2 best individuals as parents each producing 6 offspring
  - reinsertion rate 1 (`Selection.ReinsertionRate`=1.0).
- recombination algorithm: no recombination (`Recombination.Name`='recnone'),
- mutation algorithm:
  - evolution strategy mutation 2 (`Mutation.Name`='`mutes2`'),
  - mutation range 0.01 (`Mutation.Range`=[0.01]).

`tbx3comp` (switches competition on, default competition options are employed):

- competition: use competition (`Competition.Do`=1).
- migration: use migration (`Migration.Do`=1).

`tbx3perm` (ordering/permutation problem, standard operators for permutation problems):

- permutation variables: `VariableFormat`=5 (the EA just exchanges/permutes the variables, after initialization (by `initpop`/`initpp`) the values are not changed),
- recombination algorithm: `Recombination.Name`='`recpm`',
- mutation algorithm: `Mutation.Name`='`mutswap`',

All these ToolBoX functions are directly available inside the GEATbx. They define a number of algorithms that should help solve many problems. Simultaneously they serve as template to write own ToolBoX functions or support the definition of special algorithms.

1.2 Option handling with geaoptset.m

The main file for all option handling (GEATbx version 3.x) is `geaoptset`. This file is similar to simset (but can handle multi-level options and checks the validity of the options).

The function `geaoptset` may be used in multiple ways. All these possibilities will be explained in the following paragraphs:

- info on available options,
- define and add single or multiple options,
- merge option structures,
- check the validity of options.

Default Options

When calling `geaoptset` without any input or output parameter a long table containing all possible options and their default values, value ranges, and types is printed on the screen.

Properties of GEA Toolbox and their possible and default settings:

Tab. 1-1: Table of default option settings (defined in `geaoptset`)

```
>> geaoptset

Property Name          Type             min. Range max.  Numb. of para.   Default value
 NumberSubpopulation   positive integer   1    Inf       just one         4
 NumberIndividuals     positive integer   1    Inf       num. subpop      25 20 15 30
 VariableFormat        positive integer   0      5       just one         0
 Selection.Name        string           NaN    NaN       num. subpop     {1}  selsus
 Selection.Pressure    positive scalar    0    Inf       num. subpop      1.7
...
```

If you just want to get the default set of options, call `geaoptset` with one output parameter. This returns a structure containing all possible options with their default settings:

```
>> DefOptions = geaoptset;
```

The GEATbx contains a function to display (nearly) every variable in a pretty way: `prprintf`. Using this function the full options structure can be displayed. (The functionality of `prprintf` can be used with any other structure / cell array / matrix / scalar as well.)

Tab. 1-2: Table of default option settings (output from `prprintf`)

```
>> prprintf(DefOptions)     % Some of the options are deleted
ans =
     NumberSubpopulation  :  4
     NumberIndividuals    :  25   20   15   30
     VariableFormat       :  0
     Selection
          Name               :  {1}  selsus
          Pressure           :  1.7
          RankingMethod      :  0
          GenerationGap      :  0.9
     Recombination
          Name  :  {1}  recdis
          Rate  :  1
     Mutation
          Name       :  {1}  mutreal
          Rate       :  1
          Range      :  0.1
```

```
          Precision  :  12
     Migration
          Do         :  1
          Interval   :  20
          Rate       :  0.1
... and so on
```

Define and/or Add Options

tbd.

```
GeaOpt = geaoptset(GeaOpt, 'NumberSubpopulation', 2);
```

Merge Option Structures

tbd.

Check Validity of Option Structures

To check an options structure for validity call the utility function `geaoptset` with just one argument (the options structure to check).

```
GeaOpt = geaoptset(GeaOpt);
```

During the check of the parameters the following things are done:

- check the type of the parameters (string or scalar, integer or real number),
- check the boundaries of the parameters,
- check single/multi strategy parameters,
- expand multi strategy parameters to the number of subpopulations,
- check special parameters
- include missing parameters.

At the end of the check the returned options structure is full and valid and includes all previously provided parameters.

1.3 Examples of option settings

tbd.

1.4 Status and result output during optimization

The GEATbx produces extensive status and result reports during an optimization. By default text information is written to the Matlab command window (status information). This information may be saved additionally into an text file for later retrieval and analysis (default is, this information not to save).

The option `Output.TextInterval` controls if and how often status information is displayed on the screen. By default this option has the value 1. Thus, status information is printed every generation. Higher values for `Output.TextInterval` define, that less often status information is printed on the screen. A value of 5 defines, that status information is displayed every 5 generations.

The option `Output.SaveTextInterval` controls if and how often the status information is saved into an text file. By default, this option is switched off (option value is 0). By setting this option to a value of 1 or higher status information is saved to a text file every specified generation (the name of the file is defined in the option `Output.SaveTextFileName`).

The contents of the status information printed on screen and saved into a text file is (basically) identical. By using the options `Output.TextExclude*` and `Output.SaveTextExclude*` the contents of the status information can be adjusted.

At the beginning of an optimization the used options of the optimization are displayed in a formatted form. This information is called header. Only the used options are displayed and some comments may be added. Thus, every used parameter is fully documented - even when this setting was not directly set by the user but taken from one of the predefined ToolBoX functions or the default options set. Figure 1-1 contains an example of the displayed header information (from running `demofun1`).

Fig. 1-1: Output of all used options at the beginning of an optimization (Header of status information)

```
Evolutionary Optimization

Objective function:  objfun1       Date: 05-Mar-2000     Time: 21:46:12
   number of variables:   10
   boundaries of variables:        -512
                                    512

Evolutionary algorithm parameters:
   subpopulations =      4       individuals =     25   (at start per subpopulation)
   termination    1:  max. generations = 400;
   variable format =    0   (real values - phenotype == genotype)
   selection
         function = selsus
         pressure =    1.7
         gen. gap =    0.9
      reinsertion
            rate =      1
          method =      2
   recombination
         name = recdis  recdis  reclin  recdis
         rate =      1
   mutation
           name = mutreal
           rate =      1
          range =    0.1    0.01   0.001  0.0001
      precision =     16
   regional model
      migration
            rate =   0.1    interval =   20    topology =    0    selection =  1
   output
      results on screen every  5  generation
      grafical display of results every  10  generation
         method = 111111
         style  = 614143
      results into text file every  5  generation
          file name =  res_beasv_objfun1_var_1_01.txt
```

During the optimization status information is displayed in tabular format, one line per generation (or as often as defined in `Output.TextInterval`). The table header provides a short description of the information contained in the status table. The run-status information contains:

- number of current generation,
- number of objective function calls,
- best objective value found so far (one value for single-objective optimization, multiple value for the multi-objective case),
- variables of the best individual (not displayed on screen by default - may be switched on as well),
- position of the subpopulations (applies only when using multiple subpopulations),
- size of the subpopulations (applies only when using competition between subpopulations),
- information on the termination criteria (how much each of the used termination criteria is full-filled),
- calculation time (time for one generation, overall time for the optimization until now).

The last 5 types of information may be switched on or off. Please check the description of the options `Output.TextExclude*` and `Output.SaveTextExclude*`.

According to the above header information, every 5 generations one line of status information is displayed.

Fig. 1-2: Status information displayed in command window during optimization (some lines removed)

```
Generation   f-Count     Obj. Function       Pos of subpopulations / size                  Term: 1       Time: cpu/gen,  full
   5.          451            40519         3   2   1   4      31     23     23     23      [   1.25%]     (  0.00min  00:00:02)
  10.          888            30140         2   3   1   4      28     21     30     21      [   2.50%]     (  0.00min  00:00:03)
  15.         1331            12935         3   1   2   4      26     19     36     19      [   3.75%]     (  0.00min  00:00:06)
  20.         1776           8572.8         3   1   2   4      22     17     44     17      [   5.00%]     (  0.00min  00:00:06)
  25.         2217           4910.8         3   2   1   4      20     24     40     16      [   6.25%]     (  0.00min  00:00:08)
```

```
  30.         2660          3644.4         3   1   2   4      18     22     45     15      [   7.50%]     (  0.00min  00:00:09)
  35.         3100          2414.4         2   1   3   4      17     28     41     14      [   8.75%]     (  0.00min  00:00:11)
  40.         3540          1210.6         2   1   3   4      15     39     34     12      [  10.00%]     (  0.00min  00:00:11)
  45.         3979          678.95         3   1   2   4      14     44     31     11      [  11.25%]     (  0.00min  00:00:13)
  50.         4418          208.31         3   1   2   4      13     49     28     10      [  12.50%]     (  0.00min  00:00:14)

 200.        17636       1.1752e-005       4   2   3   1       5      5     18     72      [  50.00%]     (  0.00min  00:00:54)
 205.        18076       8.6381e-006       4   3   2   1       5      5     17     73      [  51.25%]     (  0.00min  00:00:56)
 210.        18516       8.6179e-006       4   3   2   1       5      5     16     74      [  52.50%]     (  0.00min  00:00:57)
 215.        18956       5.6262e-006       1   4   3   2       5      5     15     75      [  53.75%]     (  0.00min  00:00:58)
 220.        19396       4.2513e-006       3   4   1   2       5      5     13     77      [  55.00%]     (  0.00min  00:00:59)

 370.        32604       4.4879e-008       3   4   2   1       5      5     13     77      [  92.50%]     (  0.00min  00:01:38)
 375.        33044       3.9711e-008       4   3   1   2       5      5     12     78      [  93.75%]     (  0.00min  00:01:40)
 380.        33484       3.4701e-008       2   4   1   3       5      5     10     80      [  95.00%]     (  0.00min  00:01:41)
 385.        33928       3.2459e-008       3   4   2   1       5      5      9     81      [  96.25%]     (  0.00min  00:01:42)
 390.        34368       3.1356e-008       3   4   2   1       5      5      8     82      [  97.50%]     (  0.00min  00:01:43)
 395.        34808       1.4482e-008       3   4   2   1       5      5      7     83      [  98.75%]     (  0.00min  00:01:45)
 400.        35248       1.4482e-008       4   3   2   1       5      5      5     85      [ 100.00%]     (  0.00min  00:01:46)
```

At the end of the optimization (when any one of the termination criteria is full-filled) result information is displayed on the screen (and saved into a text file if specified). The result information contains:

- which termination criteria was full-filled (here: max. number of generations),
- how long the optimization run (number of generations and time),
- best objective value found (one value for the single-objective case and multiple values for the multi-objective case) and when this result was found (number of generation),
- variable values of the best individual.

Fig. 1-3: Result information displayed in command window at the end of the optimization

```
End of optimization: max. generations (400 generations; 1.78 cpu minutes / 1.78 time minutes)
Best Objective value:  1.44825e-008   in Generation 393
Best Individual:   -2.734e-005 -1.4079e-005   2.419e-005 -2.9976e-005 -2.8788e-005  9.4641e-005 -1.3272e-005  3.2009e-005  2.7127e-005  -
1.8188e-005
```

All the status information together (header, status of run and result of optimization) provide a detailed overview of a complete optimization run with as much information as needed normally. When saving the information into a text file as well (`Output.SaveTextInterval`) it can be used for later (off-line) analysis. At the same time this textual information provides a good documentation of an optimization run even readable years later.

1.5 Description of Options

The following Chapters explain all available options of the GEATbx version 3.x in detail. Each of the following Chapters describes the options of a group (General, Recombination, Mutation, ...).

Only a few of the available options are used at a time by the user. Most of these options are only needed for very special cases. The GEATbx was and is used for many different tasks - and some of the options are only needed then. All options not specially defined by the user are preset and contain useful values.

Each option description contains:

- a short description of the option,
- the default value and type of the option (defined in `geaoptset`),
- the meaning of the different values or the available parameter values (i.e. function names for recombination or which value means switch option on or off),
- short examples of setting the option (and possibly related options),
- special tips for the application of the option,
- which other options correspond with the option (directly),
- the option numbers in GEATbx version 1.9x and 2.x.

2 General options

NumberSubpopulation

This option defines the number of subpopulations to use.

- Default value: 4
- Type: positive integer in [1, Inf]
- Example - use 2 subpopulations:
  ```
  GeaOpt = geaoptset(GeaOpt, 'NumberSubpopulation', 2);
  ```
- The setting of this parameter corresponds with `NumberIndividuals`. The number of all individuals in the population is the sum of all individuals in the subpopulations. For instance, 4 subpopulations with 20 individuals each produce a population of 80 individuals.
- When only one subpopulation is defined the population is handled as one (panmictic) population. Two and more subpopulations define the use of the regional population model (also called island model or coarse-grained model) employing migration between subpopulations (see `Migration.Do`).
- The setting of this option corresponds with `NumberIndividuals`.
- Option number in previous versions: 21 (2.x and 1.x)

NumberIndividuals

This option defines the number of individuals per subpopulation to use.

- Default value: [25, 20, 15, 30]
- Type: positive integer in [1, Inf]
- The number of individuals can be the same for all subpopulations or different. Thus, special values for every subpopulation can be defined (multi strategy support).
- Example - set number of individuals in all subpopulations to 20:
  ```
  GeaOpt = geaoptset(GeaOpt, 'NumberIndividuals', 20);
  ```
- Example - set number of individuals in each subpopulation to a specific number:
  ```
  GeaOpt = geaoptset(GeaOpt, 'NumberIndividuals', [20,15,25,35]);
  ```
- A population/subpopulation should have at least 15 individuals. The larger and more complex a problem is the higher the number of individuals ought to be.
- The setting of this option corresponds with `NumberSubpopulation`.
- Option number in previous versions: 20 (2.x and 1.x)

VariableFormat

This option defines the format (real, integer, binary, ordering) of the variables and the conversion (done by `bindecod`) between this format and the internally used representation of the variables. All conversions are handled inside the main function `geamain2` (fully transparent to the user):

- Default value: 0
- Type: integer in [1, 5]
- Depending on the setting a conversion between internal representation (genotype) and the representation of the variables (phenotype) is carried out:
  - 0: real values; no conversion (the EA works on real values)
  - 1: real values; the EA works on binary values (conversion from binary to real)
  - 2: integer values; no conversion (the EA works on integer values)
  - 3: integer values; conversion from binary to integer (the EA works on binary values)
  - 4: binary values; no conversion (the EA works on binary values)
  - 5: ordering / permutation / scheduling problems, any representation of variables; no conversion
- Example - the problem employs integer valued variables and the EA should work on binary values, (convert the internal binary representation to the external integer representation):
  ```
  GeaOpt = geaoptset(GeaOpt, 'VariableFormat', 2);
  ```
- The setting of this option corresponds with the recombination and mutation options. Select appropriate operators and their corresponding options: `Recombination.Name`, `Mutation.Name`.

- Depending on the internal representation of the variables the appropriate initialization functions is called inside `geamain2`/`initpop`:
  - Real values (0): initialization by `initrp`
  - Integer values (2): initialization by `initip`
  - Binary values (1, 3, 4): initialization by `initbp`
  - Ordering /permutation / scheduling (5): initialization by `initpp`
- Option number in previous versions: 19 (2.x and 1.x)

3 Selection options

Selection.Name

This option contains the name of the selection function (name of m-file).

- Default value: selsus
- Type: string or cell array of strings; multi strategy support
- Available selection functions:
  - selsus stochastic universal sampling
  - seltrunc truncation selection
  - seltour tournament selection
  - selrws roulette wheel selection
  - sellocal local selection (local population model)
  - Any other selection function may be used directly; set Selection.Name to the name of the m-file.
- Example - use tournament selection seltour:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.Name', 'seltour');
  ```
- Example - use tournament selection seltour for first subpopulation and truncation selection seltrunc for second subpopulation:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.Name', ...
                             {'seltour', 'seltrunc'});
  ```
- Option number in previous versions: 5 (2.x and 1.x), the global variable GLOBAL_SELECTIONFUN is no longer necessary to define special selection functions.

Selection.Pressure

The value of selection pressure determines the fitness assignment and is used by the ranking algorithm. Fitness assignment is always done by ranking (there is no function for proportional fitness assignment).

- Default value: 1.7
- Type: positive scalar in [1, 2] or [1, Inf]; multi strategy support
- Example - set selection pressure for all subpopulations to 1.9:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.Pressure', 1.9);
  ```
- Inside every selection function (Selection.Name) the selection pressure is converted into the algorithm specific parameter (for instance tournament size in tournament selection seltour or truncation threshold in truncation selection seltrunc). See the function/operator descriptions for the used conversions.
- The setting of this option determines/corresponds with the employed ranking method. This behavior may be changed with the option Selection.RankingMethod:
  - Selection.Pressure in [1, 2]: use linear ranking
  - Selection.Pressure > 2: use non-linear ranking
- Option number in previous versions: 26 (2.x) or 23 (1.x)

Selection.GenerationGap

This option defines the generation gap, the fraction of the population to be reproduced every generation.

- Default value: 0.9
- Type: scalar in [0, Inf]; multi strategy support
- Example - set generation gap to 70%:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.GenerationGap', 0.7);
  ```
- Three different ranges of values for the generation gap exist:
  - Selection.GenerationGap < 1.0: less offspring than individuals in population are produced. Thus, some individuals of the population survive. This is identical to elitest selection.
  - Selection.GenerationGap > 1.0: more offspring than individuals in population are produced. Not all offspring will be inserted into the population.

  - Selection.GenerationGap very small (<0.2): only a few offspring are produced. This is identical to steady state algorithms.
- The setting of this option corresponds with Selection.ReinsertionRate.
- Option number in previous versions: 22 (2.x and 1.x)

Selection.ReinsertionRate

This option defines how much parents are replaced by the produced offspring. The reinsertion rate is only active when the Selection.GenerationGap is larger than the reinsertion rate.

- Default value: 1
- Type: scalar in [0, 1]; multi strategy support
- A reinsertion rate of 1 means that all parents may be replaced by offsprings (provided the generation gap is 1 or larger). A value smaller than 1 assures a few parents are not replaced by offspring. If less offspring are produced (determined by Selection.GenerationGap) than could be reinserted, only the produced offspring are inserted into the population. The reinsertion rate is used inside the reinsertion function reins.
- Example - set reinsertion rate to 0.8:
```
GeaOpt = geaoptset(GeaOpt, 'Selection.ReinsertionRate', 0.8);
```
- The setting of this option corresponds with Selection.GenerationGap.
- The reinsertion rate is useful when Selection.GenerationGap is larger than 1 (more offspring than parents are produced) and not all parents should be replaced. With a reinsertion rate of smaller than 1 an 'elitest selection' can be implemented.
- Option number in previous versions: 23 (2.x), not available/internal option (1.x)

Selection.RankingMethod

This option sets the employed ranking method (fitness assignment). Employ this option only when the ranking method should be set to non-linear ranking for a Selection.Pressure <= 2.

- Default value: 0
- Type: integer in {0, 1}; multi strategy support
- Example - set ranking method to non-linear ranking for a selection pressure of 1.3:
```
GeaOpt = geaoptset(GeaOpt, 'Selection.Pressure', 1.3, ...
                           'Selection.RankingMethod', 1);
```
- The setting of this option corresponds with Selection.Pressure.
- Option number in previous versions: not available/internal option (2.x and 1.x)

Selection.RankingMultiobj

This option defines the use of multi-objective ranking (fitness assignment). Employ this option only when the objective function uses/returns multiple objective values and multi-objective ranking should be employed.

- Default value: 0
- Type: integer in [0, Inf]; multi strategy support
- Example - switch multi-objective ranking on:
```
GeaOpt = geaoptset(GeaOpt, 'Selection.RankingMultiobj', 1);
```
- The multi-objective ranking is done inside the ranking function. All other ranking parameters apply as well (selection pressure, ranking method). Multi-objective ranking means, that the sorting/ranking of the individuals is done multi-objective (using pareto ranking and goal attainment) instead of the simple sorting of single-objective individuals. An example for multi-objective ranking and the visualization of such functions is provided in demomop.
- This option corresponds with System.ObjFunGoals.
- Option number in previous versions: not available (2.x and 1.x)

Selection.ReinsertionMethod

This option sets the employed reinsertion method (selection of offspring for reinsertion in reins).

- Default value: 2
- Type: integer in [0, 6]; multi strategy support
- Reinsertion methods for the global and regional population model (reinsreg), the option Selection.ReinsertionRate is employed:
  - 0, 1: uniform at random selection
  - >=2: fitness based selection

- Reinsertion methods for the local model (reinsreg), the Selection.ReinsertionRate is not used:
  - 0: all offspring are reinserted, neighbors are replaced randomly
  - 1: all offspring are reinserted, weakest neighbors are replaced
  - 2: only offspring fitter than weakest neighbor are reinserted, weakest neighbors are replaced
  - 3: only offspring fitter than weakest neighbor are reinserted, parents are replaced
  - 4: only offspring fitter than weakest neighbor are reinserted, neighbors are replaced randomly
  - 5: only offspring fitter than parents are reinserted, parents are replaced
  - 6: all offspring are reinserted, parents are replaced
- Example - set reinsertion method to uniform at random (global/regional population model):
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.ReinsertionMethod', 0);
  ```
- The setting of this option corresponds with Selection.Name and Selection.ReinsertionRate.
- Option number in previous versions: 30 (2.x), not available/internal option (1.x)

Selection.LocalDimension

This option determines the number of dimensions of the population when using the local population model (the use of the local population model is defined by using sellocal as selection algorithm (Selection.Name) for the respective subpopulation).

- Default value: 0
- Type: integer in [0, Inf]; multi strategy support
- 0: number of dimensions is calculated depending on NumberIndividuals (see comploc),
  1: 1-D - linear,
  2: 2-D - grid,
  3: 3-D - ... (The number of dimensions of the local model could be as large as sensible.)
  The number of individuals in every dimension is calculated inside comploc. Special values may be defined by setting Selection.LocalDimension=0 and the global variable GLOBAL_LOCALDIM. These values for the number of individuals in every dimension are directly used.
- Example - set number of local dimensions to 3:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.LocalDimension', 3);
  ```
- Example - set individuals per dimension directly:
  ```
  global GLOBAL_LOCALDIM;
  GLOBAL_LOCALDIM = [2, 20]; (ladder topology)
  GeaOpt = geaoptset(GeaOpt, 'Selection.LocalDimension', 0);
  ```
- The setting of this parameter corresponds with NumberIndividuals, Selection.Name, Selection.LocalTopology and Selection.LocalDistance.
- Option number in previous versions: 27 (2.x), not available (1.x)

Selection.LocalTopology

This option determines the topology of the neighborhood when using the local population model. The calculation of the neighborhood is done in comploc.

- Default value: 0
- Type: positive integer in [0, 19]; multi strategy support
- The topology defined is independent of the dimension of the neighborhood. The internal neighborhood description calculates the correct neighbors for all possible dimensions:
  - 0: ring (1-D), cross (2-D); full topology
  - 1: star (2-D); full topology
  - 2: inclined star (2-D); full topology
  - 8: all neighbors within a radius smaller than Selection.LocalDistance; full topology
  - 9: as many neighbors as defined in Selection.LocalDistance; full topology
  - +10: the same as all previous options above with half topology
  - i.e., 11: half star (star with half topology)
- Example - set topology of local neighborhood to full cross for first and half star for second subpopulation:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Selection.LocalTopology', [0, 11]);
  ```
- The setting of this option corresponds with Selection.Name, Selection.LocalDimension and Selection.LocalDistance.

- Option number in previous versions: 28 (2.x), not available (1.x)

Selection.LocalDistance

This option determines the largest distance to neighbors or the number of neighbors when using the local population model, depending on Selection.LocalTopology.

- Default value: 1
- Type: positive integer in [1, Inf]; multi strategy support
- Example - set local topology to full star and distance to 3:

```
GeaOpt = geaoptset(GeaOpt, 'Selection.LocalTopology', 1, ...
                           'Selection.LocalDistance', 3);
```

- The setting of this option corresponds with Selection.Name and Selection.LocalTopology.
- Option number in previous versions: 29 (2.x), not available (1.x)

4 Recombination options

Recombination.Name

This option contains the name of the recombination function (name of m-file). The possible recombination operators depend on the internal representation of the variables (VariableFormat).

- Default value: recdis
- Type: string or cell array of strings; multi strategy support
- Available recombination functions:
  - recdis: Discrete recombination (all representations)
  - recint: Intermediate recombination (real valued representation)
  - reclin: Line recombination (real valued representation)
  - reclinex: Extended line recombination (real valued representation)
  - recdp: Double-point crossover (binary valued representation)
  - recdprs: Double-point reduced surrogate crossover (binary valued representation)
  - recsp: Single-point crossover (binary valued representation)
  - recsprs: Single-point reduced surrogate crossover (binary valued representation)
  - recsh: Shuffle crossover (binary valued representation)
  - recshrs: Shuffle reduced surrogate crossover (binary valued representation)
  - recgp: Generalized position recombination (ordering/permutation representation)
  - recpm: Partial matching recombination (ordering/permutation representation)
  - recnone: no recombination (internal dummy function in recombin), parents are not recombined
- Any other recombination function may be used directly, just set Recombination.Name to the name of the m-file.
- Example - use double point crossover recdp (variables of individuals must be binary):
  ```
  GeaOpt = geaoptset(GeaOpt, 'Recombination.Name', 'recdp');
  ```
- Example - use discrete recombination recdis for the first and line recombination reclin for the second subpopulation:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Recombination.Name', {'recdis','reclin'});
  ```
- The setting of this option corresponds with VariableFormat.
- Option number in previous versions: 7 (2.x and 1.x), the global variable GLOBAL_RECOMBINFUN is no longer necessary to define special recombination functions.

Recombination.Rate

This option defines the recombination rate.

- Default value: 1
- Type: scalar in [0, 1]; multi strategy support
- Example - set the recombination rate to 0.7 (often used with binary representation in genetic algorithms):
  ```
  GeaOpt = geaoptset(GeaOpt, 'Recombination.Rate', 0.7);
  ```
- Option number in previous versions: 12 (2.x), not available/internal option (1.x)

5 Mutation options

Mutation.Name

This option contains the name of the mutation function (name of m-file). The possible mutation operators depend on the internal representation of the variables (Mutation.Name).

- Default value: mutreal
- Type: string or cell array of strings; multi strategy support
- Available recombination functions:
  - mutreal: real mutation (real valued representation)
  - mutint: integer mutation (integer valued representation)
  - mutbin: binary mutation (binary valued representation)
  - mutswap: swap mutation (ordering/permutation representation)
  - mutinvert: invert mutation (ordering/permutation representation)
  - mutmove: move mutation (ordering/permutation representation)
  - mutexch: exchange mutation (ordering/permutation representation)
  - mutes1: Evolutionary strategy 1 (real valued representation)
  - mutes2: Evolutionary strategy 2 (real valued representation)
  - mutrandreal: random creation of individuals for random search (real valued representation)
  - mutrandint: random creation of individuals for random search (integer valued representation)
  - mutrandbin: random creation of individuals for random search (binary valued representation)
  - mutrandperm: random creation of individuals for random search (permutation representation)
  - mutnone: no mutation (internal dummy function in mutate), parents are not mutated
- Any other mutation function may be used directly, just set Mutation.Name to the name of the m-file.
- Example - use binary mutation mutbin (variables of individuals must be binary):
```
GeaOpt = geaoptset(GeaOpt, 'Mutation.Name', 'mutbin');
```
- Example - perform random search by random creation of individuals by „mutation“ (no recombination):
```
GeaOpt = geaoptset(GeaOpt, 'Recombination.Name', 'recnone', ...
                           'Mutation.Name', 'mutrandreal'});
```
- The setting of this option corresponds with VariableFormat.
- Option number in previous versions: 6 (2.x and 1.x), the global variable GLOBAL_MUTATIONFUN is no longer necessary to define special mutation functions.

Mutation.Rate

This option defines the mutation rate. Internally, the defined mutation rate is divided by the number of variables of the individuals. Thus, the defined mutation rate is just the factor of how many variables per individual are mutated (and not the internally used value).

- Default value: 1
- Type: scalar in [0, 1]; multi strategy support
- A value of Mutation.Rate=1 means that on the average 1 variable per individual is mutated, a value of 4 mutates 4 variables per individual (on average) and a value of 0.2 only 1 variable per 5 individuals.
- Example - set the mutation rate to 2 (higher mutation rate than default):
```
GeaOpt = geaoptset(GeaOpt, 'Mutation.Rate', 2);
```
- Option number in previous versions: 11 (2.x), not available/internal option (1.x)

Mutation.Range

  - This option defines the range of the mutation steps for every variable depending on the size of the domain of the respective variable or the definition of the respective operators. This option is not necessary for binary valued representations (mutbin).
- Default value: 0.1
- Type: scalar in [1, 0]; multi strategy support

- Mutation range is also used for defining the initial step sizes of evolution strategy mutation (`mutes1` and `mutes2`). When mutation range is 1.0, the initial step sizes will be 0.01 times domain of variable, for mutation range of 0.2, the initial step sizes will be 0.002 times domain of variable.
- Example - set the mutation range of 4 subpopulations (defines rough / standard / fine / very fine search):
  ```
  GeaOpt = geaoptset(GeaOpt, 'Mutation.Range', [0.3 0.1 0.03 0.01]);
  ```
- Option number in previous versions: 24 (2.x), not available/internal option (1.x)

Mutation.Precision

This option defines the precision of the mutation steps depending on the mutation range. This option is not necessary for binary valued representations (`mutbin`).

- Default value: 12
- Type: scalar in [1, Inf]; multi strategy support
- Example - set the mutation precision for 4 subpopulations (from rough to very fine precision):
  ```
  GeaOpt = geaoptset(GeaOpt, 'Mutation.Precision', [8, 12, 16, 24]);
  ```
- Option number in previous versions: 25 (2.x), not available/internal option (1.x)

6 Migration options

Migration.Do

This option switches migration between subpopulations on or off. Migration is executed by migrate.

- Default value: 1
- Type: integer in {0, 1}
- Two variants are available:
  - 0: no migration
  - 1: do migration between subpopulations
- Example - switch migration off:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Migration.Do', 0);
  ```
- When only one subpopulation is defined (see NumberSubpopulation) no migration takes place.
- Option number in previous versions: 31 (2.x), not available/internal option (1.x)

Migration.Interval

This option defines the number of generations between successive migration. Thus, the frequency of migration is specified (how often a migration takes place). This option is often called isolation time.

- Default value: 20
- Type: positive integer in [1, Inf]
- Example - set migration interval to every 8 generations:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Migration.Interval', 8);
  ```
- Small values of Migration.Interval decrease the isolation of the individuals.
- A migration takes place (that means the function migrate is called) when the reminder of Run.Generation and Migration.Interval is zero.
- Option number in previous versions: 33 (2.x), not available/internal option (1.x)

Migration.Rate

This option defines the fraction of every population to migrate during a migration.

- Default value: 0.1
- Type: scalar in [0, 1]
- Example - set migration rate to 15%:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Migration.Rate', 0.15);
  ```
- A value of 0.15 means that 15% of the individuals of every subpopulation are copied (exported) to the migration pole. From this pole the same number of individuals is selected and migrates (is imported) to the subpopulation. It is assured that no individuals from the own subpopulation are reimported. The selection of the individuals is controlled by the option Migration.Selection
- Option number in previous versions: 32 (2.x), 26 (1.x)

Migration.Topology

This option defines the used topology of the subpopulations for migration.

- Default value: 0
- Type: integer in {0, 1, 2}
- The following variants are available:
  - 0: complete net structure (unconstrained migration)
  - 1: 1-D neighborhood structure
  - 2: 1-D ring structure
- Example - use 1-D ring structure between subpopulations:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Migration.Topology', 2);
  ```
- Option number in previous versions: 34 (2.x), 27 (1.x)

Migration.Selection

This option controls the selection of individuals for migration.

- Default value: 1
- Type: integer in {0, 1}
- Two variants are available:
  - 0: the exported individuals are selected uniform at random
  - 1: the best individuals are exported
- Example - select the individuals for migration uniform at random:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Migration.Selection', 0);
  ```
- Option number in previous versions: not available/internal option (2.x and 1.x)

7 Competition options

Competition.Do

This option switches competition between subpopulations on or off. Competition is executed by compete. Competition is an extension of the regional population model.

- Default value: 0
- Type: integer in {0, 1}
- Two variants are available:
  - 0: no competition
  - 1: do competition between subpopulations
- Example - switch competition on:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Competition.Do', 1);
  ```
- Competition without migration (Migration.Do) doesn't make sense. When only one subpopulation is defined (see NumberSubpopulation) no competition takes place.
- Option number in previous versions: 36 (2.x), not available (1.x)

Competition.Interval

This option specifies how often a competition takes place. Thus, the frequency of competition is defined.

- Default value: 4
- Type: positive integer in [1, Inf]
- Example - set competition interval to every 10 generations:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Competition.Interval', 10);
  ```
- A competition takes place (that means the function compete is called) when the reminder of Run.Generation and Competition.Interval is zero.
- Option number in previous versions: 38 (2.x), not available (1.x)

Competition.Rate

This option defines the fraction of every subpopulation to be transferred from unsuccessful subpopulations during a competition.

- Default value: 0.1
- Type: scalar in [0, 1)
- Example - set competition rate to 5%:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Competition.Rate', 0.05);
  ```
- Option number in previous versions: 37 (2.x), not available (1.x)

Competition.SubpopMinimum

This option defines the minimal number of individuals per subpopulation. When a subpopulation contains this minimal number of individuals, no further individuals are transferred to successful subpopulations.

- Default value: 5
- Type: positive integer in [1, Inf]
- Example - set minimal number of individuals to 10 individuals:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Competition.SubpopMinimum', 10);
  ```
- Option number in previous versions: 39 (2.x), not available (1.x)

8 Termination options

All the termination is handled by the function `terminat`.

`Termination.Method`

This option sets the employed methods for termination of the optimization. You may select none, one or multiple of the available termination methods.

- Default value: 1
- Type: integer in [1, Inf]
- When `Termination.Method` is 0, no termination at all takes place.
- Multiple termination methods may be used at the same time. The methods are OR combined. That means, when one of the employed termination methods is true, the optimization is finished.
- The following termination methods are available. The parameters of the termination methods are defined by specific options given in parenthesis.
  - 1: maximal number of generations (`Termination.MaxGenerations`)
  - 2: maximal computing time in minutes (`Termination.MaxTime`)
  - 3: minimal difference to defined optimum (`Termination.Diff2Optimum`)
  - 4: running mean of best objective values (`Termination.RunningMean`)
  - 5: minimal standard deviation of all current objective values (`Termination.StdObjV`)
  - 6: good worst individual/objective value (`Termination.GoodWorstObjV`)
  - 7: Phi convergence (`Termination.Phi`)
  - 8: Kappa convergence (`Termination.Kappa`)
  - 9: cluster analysis (`Termination.Cluster`)
- Two styles for the definition of multiple termination methods are available:
  - The termination methods are defined as a vector of numbers, each element of the vector specifying one employed termination method: [1 3 6].
  - The termination methods are contained in one number and each digit of the number defines one of the employed termination methods: 136.
- Example - employ termination methods maximal generations and maximal time (both styles shown):

```
GeaOpt = geaoptset(GeaOpt, 'Termination.Method', 12);
GeaOpt = geaoptset(GeaOpt, 'Termination.Method', [1 2]);
```

- Example - employ termination methods maximal generations, difference to optimum and running mean:

```
GeaOpt = geaoptset(GeaOpt, 'Termination.Method', 134);
GeaOpt = geaoptset(GeaOpt, 'Termination.Method', [1 3 4];
```

- Not all termination methods ensure, that an termination takes place (in finite time). Thus, employ always one or multiple of the reliable termination methods (`Termination.MaxGenerations`, `Termination.MaxTime`, `Termination.RunningMean`).
- Option number in previous versions: 50 (2.x), not available (1.x)

`Termination.MaxGenerations`

This option determines the maximal number of generations an optimization is run. When the specified number of generations is reached, the optimization terminates.

- Default value: 100
- Type: integer in [1, Inf]
- Example - define the maximal number of generations before termination to 234:

```
GeaOpt = geaoptset(GeaOpt, 'Termination.MaxGenerations', 234, ...
                           'Termination.Method', 1);
```

- A good starting value for maximal number of generations depending on the number of independent variables is: `200 times sqrt(n)` (`n` is the number of independent variables).
- Option number in previous versions: 51 or 14 (2.x), 14 (1.x)

Termination.MaxTime

This option determines the maximal time (in minutes) an optimization is run. When the specified time is over, the optimization terminates.

- Default value: 10
- Type: integer in [0, Inf]
- Example - define the maximal optimization time to 4.5 minutes:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.MaxTime', 4.5, ...
                           'Termination.Method', 2);
```
- The computing time is (currently) measured as real/absolute time (and not cpu time). Before a generation starts, the time is measured. If this time is longer than specified, the optimization terminates.
- Option number in previous versions: 52 (2.x), not available (1.x)

Termination.Diff2Optimum

This option terminates an optimization run, if the best objective value reached a defined value (measure of the precision required of the objective function at the solution). If the difference between the best objective value and the defined global optimum (precision of solution) is smaller than `Termination.Diff2Optimum`, the termination criteria is true and the optimization terminates.
This may be used for benchmarking different optimization algorithms (difference to known global optimum) or for termination, when a good enough value is found (problem specific). The global optimum/good enough objective value must be defined in `System.ObjFunMinimum`.

- Default value: 0.0001
- Type: scalar in [0, Inf]
- Example - define the difference to global optimum to 0.01 and the global optimum to 3.45. As soon as an objective value of 3.46 or smaller is found, the optimization terminates:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.Method', [3 1], ...
   'Termination.Diff2Optimum', 0.01, 'System.ObjFunMinimum', 3.45);
```
- This termination method is currently only implemented for single-objective optimization. An extension to the multi-objective case is possible. However, currently only the first objective value is used.
- Option number in previous versions: 53 or 3 (2.x), not available (1.x)

Termination.RunningMean

This option determines the minimal difference between the mean of the best objective values of last *RunMean* generations and the current best objective value. Internally, *RunMean* is set to 15 Generations.

- Default value: 0
- Type: integer in [0, Inf]
- Example - define the running mean to 0.01:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.RunningMean', 0.01, ...
                           'Termination.Method', 4);
```
- A value for `Termination.RunningMean` of 0 defines, that the optimization is only terminated, when for 15 (the defined value of *RunMean*) consecutive generations no better objective value at all is found. A higher value terminates the optimization, when the enhancement of the best objective value is very small only. Running Mean is one of the reliable termination criteria.
- Option number in previous versions: 55/54 (2.x), not available (1.x)

Termination.StdObjV

This option determines the minimal value of the standard deviation of the objective values of the current generation to reach before termination.

- Default value: 0.001
- Type: integer in [0, Inf]
- Example - define the minimal standard deviation of the objective values to 0.42:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.StdObjV', 0.42, ...
                           'Termination.Method', [5 1]);
```
- The standard deviation of the objective values is problem specific. The size of the termination option depends on the used evolutionary operators too. This termination option is not reliable. Use it only, when you know what you are doing!
- Option number in previous versions: 54/55 (2.x), not available (1.x)

Termination.GoodWorstObjV

This option determines the minimal difference between the objective values of the current worst and best individual.

- Default value: 0.1
- Type: integer in [0, Inf]
- Example - define the minimal difference between best and worst objective value to 11.45:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.GoodWorstObjV', 11.45, ...
                           'Termination.Method', [6 1]);
```
- The difference of the best and worst objective value is problem specific. The size of the termination option depends on the used evolutionary operators too. This termination option is not reliable. Use it only, when you know what you are doing!
- Option number in previous versions: 56 (2.x), not available (1.x)

Termination.Phi

This option determines the minimal difference between 1 and *Phi*. *Phi* is the quotient of the average objective value (of all individuals of the population) and the best objective value.

- Default value: 1e-006
- Type: integer in [0, Inf]
- Example - define the termination with *Phi* to 0.0001:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.Phi', 0.0001, ...
                           'Termination.Method', [7 1]);
```
- The value of *Phi* is problem specific. The termination option depends on the used evolutionary operators too. This termination option is not reliable. Use it only, when you know what you are doing!
- Option number in previous versions: 57 (2.x), not available (1.x)

Termination.Kappa

This option determines the minimal difference between 1 and *Kappa*. *Kappa* is a measure for the similarity of the individuals.

- Default value: 1e-006
- Type: integer in [0, Inf]
- Example - define the termination with *Kappa* to 0.00002:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.Kappa', 0.00002, ...
                           'Termination.Method', [8 1]);
```
- The value of *Kappa* is problem specific. The termination option depends on the used evolutionary operators too. This termination option is not reliable. Use it only, when you know what you are doing!
- Option number in previous versions: 58 (2.x), not available (1.x)

Termination.Cluster

This option defines the termination option for cluster analysis - only available in internal version. The calculation of the cluster termination is complex. Please look into the respective documentation.

- Default value: 0
- Type: integer in [0, Inf]
- Example - define the cluster termination option to 0.01:
```
GeaOpt = geaoptset(GeaOpt, 'Termination.Cluster', 0.01, ...
                           'Termination.Method', [9 1]);
```
- Option number in previous versions: not available (2.x and 1.x)

9 Output and Visualization options

Output.TextInterval

This option controls if any and how much text output (status information) is displayed on the screen during an optimization. A value of 0 switches text output off, a value of 1 displays information every generation (one line per generation) and higher values display information only every defined generation.

- Default value: 1
- Type: integer in [0, Inf]
- Example - set text output on screen to every 5 generations:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.TextInterval', 5);
  ```
- Example - switch text output on screen off:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.TextInterval', 0);
  ```
- Option number in previous versions: 1 (2.x and 1.x)

Output.GrafikInterval

This option controls if any and how often graphical output is presented on the screen during an optimization. The definition of the interval is similar to `Output.TextInterval`.

- Default value: 0
- Type: integer in [0, Inf]
- Example - switch graphical output every 10 generations on:
  ```
  GeaOpt = geaoptset(GeaOpt, '', 10);
  ```
- The visualized data and the style of the data is defined by `Output.GrafikMethod` and `Output.GrafikStyle.`
- Option number in previous versions: 61 (2.x), 1 (1.x)

Output.GrafikMethod

This option sets the employed methods for graphical output (here called plot mask - which graphics are plotted on screen). You may select none, one or multiple of the available methods (data sets). An overview of the available graphics/visualization methods is given under `Output.GrafikStyle`. Please look at the extensive examples in Section ??? (to be written).

- Default value: 111111
- Type: integer in [1, 1e+10]
- When `Output.GrafikMethod` is 0, the standard plot mask defined inside `resplot` is used.
- When defining a plot mask, each digit of the number defines, if the corresponding data type should be visualized:
  - 0: no display of this data type
  - 1: display this data type
  - Every position of the given number determines one plot, the first number the first data type plot, the second number the second data type plot and so on. (When excluding the first data type, use a 2 at this position.) When `Output.GrafikMethod` has fewer than the maximal digits, the remaining digits are set to zero. Currently, eight (nine) different data types can be visualized.
- Example - employ only the first two data types/methods:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.GrafikMethod', 11);
  ```
- Example - employ only the third data type/method:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.GrafikMethod', 201);
  ```
- Example - employ only the first, second, and fourth data type/method:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.GrafikMethod', 1101);
  ```
- The frequency of visualization is defined by `Output.GrafikInterval` and the styles of the displayed methods by `Output.GrafikStyle`.
- Option number in previous versions: 62 (2.x), not available (1.x)

Output.GrafikStyle

This option determines the style of the plot of each data type/method.

- Default value: 0
- Type: integer in [0, Inf]
- When Output.GrafikStyle is 0, the standard plot style defined inside resplot is used.
- Every digit of the given number determines one plot style , the first number the style of the first data type plot, the second number the style of the second data type plot and so on.
- Example - define the style for the first 3 data types/plots/methods:
```
GeaOpt = geaoptset(GeaOpt, 'Output.GrafikStyle', 244);
```
- Example - define the style of the first 4 data types/methods, but display only the second and the fourth:
```
GeaOpt = geaoptset(GeaOpt, 'Output.GrafikMethod', 2101, ...
                           'Output.GrafikStyle',  2441);
```
- The frequency of visualization is defined by Output.GrafikInterval and the methods for visualization by Output.GrafikMethod.
- Option number in previous versions: 63 (2.x), not available (1.x)
- The following list gives an short overview of the available plot methods/data types and their styles.
- 1. data type: best objective value of every generation (convergence diagram)
  - **1:** 2-D line plot of best objective value
  - **2:** 2-D line plot of best and mean objective value and standard deviation of all objective values
  - **3:** same as 1, with log scaling
  - **5:** 2-D line plot of best objective value of every subpopulation
  - **6:** same as 5, with log scaling
- 2. data type: variables of best individuals in all generations
  - **1:** 2-D (multi) line plot of variables of best individual in every generation, 1 line per variable
  - **3:** 3-D line plot of variables of best individual in every generation, 1 line per variable
  - **4:** 2-D image plot of variables of best individual in every generation, color is variable value
  - **6, 8, 9:** same as 1, 3, 4 with scaled variables (normalized inside defined boundaries)
- 3. data type: all objective values of all generations
  - **1:** 2-D (multi) line plot of all objective values of all generations, 1 line per individual
  - **3:** 3-D line plot of all objective values of all generations, 1 line per individual
  - **4:** 2-D image plot (color quilt) of all objective values of all generations, color is objective value
- 4. data type: variables of all individuals in current generation
  - **1:** 2-D line plot of all variables of all individuals in current generation
  - **2:** 2-D line plot (errorbar) of all variables of all individuals in current generation
  - **3:** 3-D line plot of all variables of all individuals in current generation
  - **4:** 2-D image plot (color quilt) of all variables of all individuals in current generation
  - **6, 7, 8, 9:** same as 1, 2, 3, 4 with scaled variables (normalized inside defined boundaries)
- 5. data type: all objective values in current generation
  - **1:** 2-D point plot (scaled to some % best) of the objective values
  - **2:** 2-D fill plot (scaled to some % best) of the objective values
  - **3:** 2-D image plot (scaled to some % best) of the objective values (2-D neighborhood)
- 6. data type: size of subpopulation
  - **1:** 2-D line plot of number of individuals in every subpopulation, 1 line per subpopulation
  - **2:** 2-D line plot of relative number of individuals in every subpopulation, 1 line per subpopulation
  - **3:** 2-D line plot of positions of subpopulations, 1 line per subpopulation
- 7. data type: distance between individuals in current generation
  - **1:** 2-D stairs plot of distance between variables of all individuals (no neighborhood)
  - **2:** 2-D stairs plot of distance between variables of neighbors (1-D neighborhood)
  - **3:** 2-D image plot of distance between variables of neighbors (2-D neighborhood)
  - **4:** 3-D surf plot of distance between variables of neighbors (2-D neighborhood)
- 8. data type: same as 1. data type but displaying all generations [best objective value of every generation]
- 9. data type: cluster analysis (dendrogram and cluster tree in separate windows) – only available in internal version
  - **1:** cluster tree of all individuals of current generation
  - **2:** dendrogram of all individuals of current generation
  - **3:** cluster tree and dendrogram of all individuals of current generation (separate windows)
  - **6, 7, 8:** same as 1, 2, 3 for each subpopulation and the whole population of current generation (many separate windows)

- 10. data type: fitness-distance scatter diagram
  - **1:** 2-D point plot of fitness distance distribution of current generation
- Inside `resplot` the validity of every plot style is checked. If data is missing or the given data isn't compatible with the plot style, this plot is omitted or another valid plot style for this data type is used.

Output.SaveTextInterval

This option controls if any and how much text output (status information) is saved into a text file during an optimization. The definition of the interval is similar to `Output.TextInterval`. The saved information is extended compared to the status information on screen (variables of the best individual are added).

- Default value: 0
- Type: integer in [0, Inf]
- Example - switch saving of text output every 10 generations on:
```
GeaOpt = geaoptset(GeaOpt, 'Output.SaveTextInterval', 10);
```
- The name of the file to save must be defined in `Output.SaveTextFileName`.
- Option number in previous versions: 41 (2.x), 28 (1.x)

Output.SaveTextFileName

This option contains the file name to save text output. An optional relative or absolute path may be added (employing Matlab conventions). If no path is defined, the current working directory is used.

- Default value: '`res_gea.txt`'
- Type: string
- Example - set the file name for text results to '`res_f4_04.txt`':
```
GeaOpt = geaoptset(GeaOpt, 'Output.SaveTextFileName','res_f4_04.txt');
```
- The text result file is extended with every new run. The output is appended at the end of the file. That means multiple runs can be saved into one file. The single runs may be differentiated by the contained starting time and date stamp at the beginning of the output of every run.
- Option number in previous versions: global variable GLOBAL_FILERESULTS, this global variable is no longer necessary to define the name of the result file.

Output.SaveBinDataInterval

This option controls if any and how often binary data is saved into a binary (mat) file during an optimization. The definition of the interval is identical to `Output.TextInterval`. The saved information contains all data from the current generation and some data regarding the entire run.

- Default value: 0
- Type: integer in [0, Inf]
- Example - switch saving of binary data every 20 generations on:
```
GeaOpt = geaoptset(GeaOpt, 'Output.SaveBinDataInterval', 20);
```
- The binary data should not be saved too often. Especially for large populations or individuals with many variables large amount of disc space and time are needed. A good value for `Output.SaveBinDataInterval` is every 10 to 20 generations.
- The name of the file to save must be defined in `Output.SaveBinDataFileName`.
- Option number in previous versions: 44 (2.x), not available (1.x)

Output.SaveBinDataFileName

This option contains the file name to save binary result data (the saved data are the same as used for the visualization with `resplot`). An optional relative or absolute path may be added (employing Matlab conventions). If no path is defined, the current working directory is used.

- Default value: '`res_gea.mat`'
- Type: string
- Example - set the file name for binary results to '`res_f4_04.mat`':
```
GeaOpt = geaoptset(GeaOpt, 'Output.SaveBinDataFileName',
                           'res_f4_04.mat');
```
- The binary result file is overwritten with every new run (set a new file name for every run to save results of multiple runs).
- Using the binary result file and `reslook` the state and progress of the optimization can be further analyzed after the optimization is finished.

- Option number in previous versions: global variable GLOBAL_FILEMAT (2.x), not available (1.x), the global variable GLOBAL_FILEMAT is no longer necessary to define the name of the binary result file.

Output.StatePlotInterval

This option controls if any and how often the special problem specific visualization function is called during an optimization. This special visualization function is called state plot function. The definition of the interval is identical to Output.TextInterval.

- Default value: 0
- Type: integer in [0, Inf]
- Example - switch display of problem specific visualization every 10 generations on:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.StatePlotInterval', 10);
  ```
- The name of the file of the state plot/special visualization function must be defined in Output.StatePlotFunction.
- Option number in previous versions: 43 (2.x), 30 (1.x)

Output.StatePlotFunction

This option contains the function name of the special problem specific visualization function.

- Default value: ''
- Type: string
- Example - set the function name of the state plot function to 'plotdopi':
  ```
  GeaOpt = geaoptset(GeaOpt, 'Output.StatePlotFunction', 'plotdopi');
  ```
- The state plot function ('plotdopi') is called very similar to the objective function ('objdopi').
  Call of objective function (Chrom contains current population, method, TSTART and TEND are additional parameters):
  ```
  [ObjVal, t, x] = objdopi(Chrom, method, TSTART, TEND)
  ```
  Call of special state plot function 'plotdopi' for 'objdopi' from inside of geamain2:
  ```
  x = plotdopi(OBJ_F, chrom, gen, varargin)
  ```
  The variable OBJ_F is a string and contains the name of the objective function ('objdopi'), chrom contains the current population sorted/ranked by fitness - the first individual is the best, the second individual the second best and so on. The variable gen contains the number of the current generation (for documentation purpose). In varargin all additional parameters are contained.
  Inside the state plot function the objective function is called to get the objective values and possible additional values (here t and x). In this example (provided as part of the GEATbx) t and x contain the states of a simulation (time vector and state vectors). These vectors are used for an extensive visualization of one individual or the comparison of multiple individuals in detail.
- Option number in previous versions: global Variable GLOBAL_STATEPLOTFUN (2.x), not available (1.x), the global variable GLOBAL_STATEPLOTFUN is no longer required to define the name of the state plot function.

Output.TextExclude* and Output.SaveTextExclude*

A number of options exist to define, which text information should be written on screen and saved to the result file. These options are very seldom used/changed. The handling of the text output is done in the function gearunstatus.

- Default value: 0
- Type: integer in {0, 1}
- Two variants are possible:
  - 0: display the corresponding information (do not exclude)
  - 1: exclude the corresponding information (no output)
- The frequency of the information output is controlled by Output.TextInterval and Output.SaveTextInterval. The Exclude*-options are extensions of these (interval) options.
- The Exclude* options exist for output on screen (Output.TextExclude) and for saving to file (Output.SaveTextExclude). The meaning of the respective options is identical. Here a short description of these options is given (names for output on screen used):
  - Output.TextExcludeHead: exclude the header information (output of options, start of optimization, header of generational output)
  - Output.TextExcludePara: exclude just the additional parameters inside the header

  - `Output.TextExcludeInd`: exclude the best individual of the population per output interval (the default value of this option is 1 for `Output.TextExclude` (no output of best individual on screen) and 0 for `Output.SaveTextInterval`)
  - `Output.TextExcludeSub`: exclude the information about size and position of subpopulations (only applicable, when multiple subpopulations and/or competition are used)
  - `Output.TextExcludeTerm`: exclude the information about the termination criteria
  - `Output.TextExcludeTime`: exclude the calculation time information (how long takes one generation, how long took the whole optimization)
- `Output.TextExcludeHead` and `Output.TextExcludePara` are used only during the start of an optimization. All other options apply to the output for each output interval (which information is contained on the line per interval).
- These options are only used for special occasions. An example are special optimizations, where only a small subset of the status/result information is needed or the available place is limited.
- Option number in previous versions: not available (2.x and 1.x).

10 Result and run time options

Run.BestObjectiveValue

This option contains the best objective value found during optimization. If the objective function returns multiple objective values per individual (multi-objective optimization), this option contains a vector.

- Default value: none (Inf)
- Type: scalar or vector in [-Inf, Inf]
- Example - get the best objective value found during optimization:
  ```
  BestResult = GeaOpt.Run.BestObjectiveValue;
  ```
- This option can be used to get the number of individuals calculated with the objective function during an optimization run. This is a good measure for the expense of an optimization.
- Option number in previous versions: 8 (2.x and 1.x)

Run.CountObjFun

This is an internal option used to save the number of objective function evaluations (every calculated/evaluated individual counts one).

- Default value: none (0)
- Type: integer in [0, Inf]
- Example - get the number of objective function calls:
  ```
  ExpenseofRun = GeaOpt.Run.CountObjFun;
  ```
- This option can be used to get the number of individuals evaluated/calculated with the objective function during an optimization run. This is a good measure for the expense of an optimization.
- Option number in previous versions: 10 (2.x and 1.x)

Run.Generation

This is an internal option used to save the current generation number.

- Default value: none (1)
- Type: integer in [0, Inf]
- Example - get the number of generations the run took:
  ```
  NoGenerations = GeaOpt.Run.Generation;
  ```
- This option can be used to get the number of generations of an optimization run.

Run.DoTerminate

This is an internal option indicating, that one of the termination options was satisfied or not. It is only used inside the main function `geamain2`.

- Default value: none (0)
- Type: integer in {0, 1}

11 Objective function options

All options in this subsection are used to define/store properties of the objective function to solve: function name of the objective function, an optional description, the boundaries of the variables, additional parameters etc. The objective function name and the boundaries of the variables can also be defined by including the appropriate parameters when calling geamain2. Then these options are set inside geamain2.

System.ObjFunFilename

This option defines the name of the objective function. The name is exactly the name of the m-file (without the extension .m).

- Default value: 'objfun1'
- Type: string
- The objective function name can also be defined by setting the first input parameter when calling geamain2 to the name of the objective function (standard calling syntax of geamain2). Then this option is set inside geamain2. The value inside the first input parameter of geamain2 has a higher priority than the value in System.ObjFunFilename.
- Example - define the name of the objective function to use:
  ```
  GeaOpt = geaoptset(GeaOpt, 'System.ObjFunFilename', 'objfun6');
  ```
- Option number in previous versions: not available as option (2.x and 1.x)

System.ObjFunVarBounds

This option defines the boundaries of the variables. The definition of these values is important for a useful optimization. The defined values span the search space of the optimization. These boundaries are hard constraints (exceptions can be defined with System.ObjFunVarBoundOut). By defining the boundaries the number of variables is given as well. This option is often referred to as VLUB (Vector of Lower and Upper Bounds).

- Default value: [ ]
- Type: 2-row matrix of scalars in (-Inf, Inf)
- The boundary matrix has two rows. The first row contains the lower variable boundaries, the second row the upper boundaries. The first column defines the boundaries for the first variable, the second for the second and so on.
- The boundaries of the variables can also be defined by setting the third input parameter when calling geamain2 to the matrix of the boundaries (standard calling syntax of geamain2). Then this option is set inside geamain2. The value inside the third input parameter of geamain2 has a higher priority than the value in System.ObjFunVarBounds. When the boundaries are defined inside System.ObjFunVarBounds, just set the third input parameter of geamain2 to [] or call geamain2 with 2 input parameters only.
- Example - define the variable boundaries of 5 variables in System.ObjFunVarBounds:
  ```
  GeaOpt = geaoptset(GeaOpt, 'System.ObjFunVarBounds', ...
                             [-1,  0,  1, 5.48, -2.1; ...
                               1,  1, 20, 5.82, -1.1]);
  [x, GeaOptOut] = geamain2('objfun8', GeaOpt, []);
  ```
- Example - define a variable with the variable boundaries, use them as third input to geamain2:
  ```
  VLUB = [-1,  0,  1, 5.48, -2.1;
            1,  1, 20, 5.82, -1.1]);
  [x, GeaOptOut] = geamain2('objfun8', GeaOpt, VLUB);
  ```
- This option must be set for every system. The GEATbx contains a mechanism to encapsulate these values inside the objective function. Please look at the description of geaobjpara or any of the provided examples (obj*.m). All provided example objective functions contain this mechanism, which is automatically used.
- Please take your time to define appropriate values for the boundaries of the variables. If you are looking for good values in the range [0.1 0.2], a definition of boundaries in the range [0, 1000] would produce a

much more difficult optimization problem. An appropriate definition of the variable boundaries is one of the most important prerequisites for the successful solution of an optimization problem.
- Option number in previous versions: not available as option (2.x and 1.x)

System.ObjFunAddPara

This option contains the additional parameters of the objective function (inside an cell array). The number of additional parameters is not limited.
- Default value: {}
- Type: cell array with elements of any type
- The additional parameters of the objective function can also be defined by providing them as additional input parameter when calling geamain2 (standard calling syntax of geamain2). Then this option is set inside geamain2. The value inside System.ObjFunAddPara has a lower priority than the values in the 5+ input parameters of geamain2.
- Example - define two additional parameters, first parameter is row vector with 3 numbers, second parameter is a string containing the word 'one'. Both parameters are contained in one cell array:
```
GeaOpt = geaoptset(GeaOpt, 'System.ObjFunAddPara', {[1 2 3], 'one'});
[x, GeaOptOut] = geamain2('objfun8', GeaOpt);
```
- Example - provide the above two additional parameters directly when calling geamain2:
```
[x, GeaOptOut] = geamain2('objfun8', GeaOpt, VLUB,[],[1 2 3],'one');
```
- Option number in previous versions: not available as option (2.x and 1.x)

System.ObjFunVarBoundOut

Using this option, the defined variable boundaries may be changed from hard boundaries to soft boundaries. A value of 0 defines a hard bound, a value of 1 a soft bound. This option is used inside chkbound to reset (or not) variables outside the defined boundaries. The initialization of variables is always done inside the defined boundaries (irrespective of the settings in System.ObjFunVarBoundOut).
- Default value: []
- Type: 2-row matrix with integers in {0, 1}, same size as System.ObjFunVarBounds
- Example - define soft boundaries for the lower bound of the second variable and the upper bound of the fourth variable. All other bounds are hard bounds:
```
GeaOpt = geaoptset(GeaOpt, 'System.ObjFunVarBoundOut',
                           [ 0, 1, 0, 0, 0;
                             0, 0, 0, 1, 0]);
```
- Option number in previous versions: not available (2.x and 1.x)

System.ObjFunGoals

This option defines the goals of multiple objective values and is used in multi-objective optimization.
- Default value: []
- Type: vector of scalars in [-Inf, Inf]
- Example - set the goals for an objective function with 4 objective values:
```
GeaOpt = geaoptset(GeaOpt, 'System.ObjFunGoals', [-1.3, 4, 2, 0]);
```
- Multi-objective optimization is a large topic on its own. Some work is still to be done, including the description of the internal working of multi-objective optimization. Examples are provided in demomop, plotmop and mobjfonseca1.
- This option corresponds with Selection.RankingMultiobj.
- Option number in previous versions: not available (2.x and 1.x)

System.ObjFunMinimum

This option defines the/one known minimum of the objective function (best objective value). This option is used for termination (when a given objective value is reached).
- Default value: -Inf
- Type: scalar in [-Inf, Inf]
- Example - define the minimal objective value of the objective function to 0:
```
GeaOpt = geaoptset(GeaOpt, 'System.ObjFunMinimum', 0);
```
- Option number in previous versions: not available as option (2.x and 1.x), internal in objective function

System.ObjFunDescription

This option defines a textual description of the objective function. Currently, this option is not directly used inside the GEATbx.

- Default value: 'Objfun descr.'
- Type: string
- Example - include a description of the objective function (here for objfun6):
```
GeaOpt = geaoptset(GeaOpt, 'System.ObjFunDescription',
                           'RASTRIGINs Function 6');
```
- Option number in previous versions: not available as option (2.x and 1.x), internal in objective function

12 Special initialization options

Aside from the standard initialization (uniform at random) the user may initialize a number of individuals and provide the resulting population matrix as fourth input parameter to `geamain2`. An application-specific initialization gives the chance of incorporating application-specific knowledge. The number of individuals in the initial population can be anything from 1 to Inf, only the number of variables must be correct (the same as defined in `System.ObjFunVarBounds`). All adjustments are done in `initpop` automatically using the following options.

The preinitialized population is send to `initpop`. The following options control:

- how many of these preinitialized individuals are used unchanged (`Special.InitPresetKeep`),
- how many individuals are created uniform at random (`Special.InitUniformCreate`),
- how much further individuals are randomized around the preinitialized individuals (normal randomization). The number of normally randomized individuals is the remainder of the above two proportions and the size of the population.

The function `initpop` contains some examples (***help initpop***). The format of the variables (real, integer, binary, permutation variables - `VariableFormat`) is always considered.

`Special.InitPresetKeep`

This option controls how many preinitialized individuals (percentage of population size) should be kept unchanged in the population. If less preinitialized individuals are provided, all these individuals are kept unchanged in initial population. If more individuals are provided, only the defined percentage is kept in the initial population (individuals are selected uniform at random).

- Default value: 0.5
- Type: scalar in [0, 1]
- The following variants are available:
  - 0: keep none of the preinitialized individuals unchanged
  - > 0: scalar (percentage of population size) how many individuals from the initial population to copy to population of the first generation
  - example 0.2: keep not more than 20% of provided individuals in population
- Example - set percentage of unchanged preinitialized individuals to not more than 80%:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Special.InitPresetKeep', 0.8);
  ```
- This option is only applied, when an initial population is provided. This option corresponds with `Special.InitPresetRand` and `Special.InitUniformCreate`.
- Option number in previous versions: not available (2.x and 1.x)

`Special.InitUniformCreate`

This option controls how many individuals are created uniform at random (standard initialization, uniform at random in the domain of the variables - `System.ObjFunVarBounds`). The proportion of uniform individuals has an lower priority than the proportion of unchanged individuals (`Special.InitPresetKeep`). If the number of missing individuals is smaller than defined in `Special.InitUniformCreate`, only the remaining proportion is created uniform at random.

- Default value: 0.5
- Type: scalar in [0, 1]
- Example - set the percentage of uniformly created individuals to 40%:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Special.InitUniformCreate', 0.4);
  ```
- This option is only applied, when an initial population is provided. This option corresponds with `Special.InitPresetKeep` and `Special.InitPresetRand`.
- Option number in previous versions: not available (2.x and 1.x)

`Special.InitPresetRand`

This option controls how much preinitialized individuals are normally randomized. The number of normally randomized individuals is the remainder after keeping individuals unchanged (`Special.InitPresetKeep`)

and creating individuals uniform at random (`Special.InitUniformCreate`).
The normally randomized individuals are randomized around preinitialized individuals. This method is called innoculation. The level of randomization is such, that 1.0 corresponds to the domain of the search space. Nevertheless, the variables of the randomized individuals may be outside the defined boundaries. Currently, the variables are not reset to the boundaries (this is a feature).

- Default value: 0.25
- Type: scalar in [0, 1]
- Example - set randomization of provided individuals to 0.5:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Special.InitPresetRand', 0.5);
  ```
- This option is only applied, when an initial population is provided. This option corresponds with `Special.InitPresetKeep` and `Special.InitUniformCreate`.
- Option number in previous versions: not available (2.x and 1.x)

The following two options (`Special.InitDo` and `Special.InitFunction`) are obsolete. You may initialize a number of individuals and provide the resulting population matrix as fourth input parameter to `geamain2`. This new way (version 3.x) is much more flexible. Nevertheless, for a limited time the use of an application-specific initialization function inside `geamain2` is supported.

Special.InitDo

This option switches initialization with an application-specific initialization function on or off.

- Default value: 0
- Type: integer in {0, 1}
- Two variants are available:
  - 0: do not use a special initialization function
  - 1: use a special initialization function
- Example - switch use of special initialization function on:
  ```
  GeaOpt = geaoptset(GeaOpt, 'Special.InitDo', 1);
  ```
- Special feature: if `Special.InitDo` is larger than 10, a graphical representation of the initial population is plotted. This graphic may be used to test the special initialization. To use this feature, provide an initial population (fourth input parameter to `geamain2` or by employing a special initialization function) and set `Special.InitDo` to 11 or higher. When the initial population is provided as fourth input parameter, `Special.InitFunction` should be set to an empty string.
- Example - provide some individuals for special initialization and visualize the full initial population:
  ```
  PopInit = [ 1 2 3 4 5; 10 9 8 7 6]; % Defines two initial individuals
  GeaOpt = geaoptset(GeaOpt, 'Special.InitDo', 11, ...
                             'Special.InitFunction', '');
  [x, GeaOptOut] = geamain2('objfun8', GeaOpt, VLUB, PopInit);
  ```
- Option number in previous versions: 42 (2.x), 29 (1.x)

Special.InitFunction

This option contains the function name of the application specific initialization function.

- Default value: ''
- Type: string
- Example - set the function name of the special initialization function to '`initdopi`':
  ```
  GeaOpt = geaoptset(GeaOpt, 'Special.InitFunction', 'initdopi');
  ```
- The special initialization function ('`initdopi`') is called very similar to the objective function ('`objdopi`').
  Call of objective function (`Chrom` contains current population, `method`, `TSTART` and `TEND` are additional parameters):
  ```
  [ObjVal, t, x] = objdopi(Chrom, method, TSTART, TEND)
  ```
  Call of special initialization function '`initdopi`' for '`objdopi`' from inside of `geamain2`:
  ```
  NewPop = initdopi(Nind, VLUB, gen, varargin)
  ```
  The variable `Nind` contains the number of individuals needed, `VLUB` the boundaries of the variables. In `varargin` all additional parameters are included (cell array with `method`, `TSTART` and `TEND`). The special initialization function may not produce as many individuals as needed. If the number doesn't match `Nind`, the standard initialization function `initpop` is called and produces the missing individuals or deletes some individuals according to the special initialization options described above.

- Option number in previous versions: global Variable GLOBAL_INITFUN (2.x), not available (1.x), the global variable GLOBAL_INITFUN is no longer necessary to define the name of the special initialization function.

Special.CollectBest.Interval

This option controls if any and how much good/best individuals are collected during an optimization (see colbestind). This collection is separate from the saving of text results (Output.SaveBinDataInterval) or binary data (Output.SaveBinDataInterval). The collected individuals are compared for being not identical according to the criteria in Special.CollectBest.Compare. When an individual is selected, the objective value(s) and the variables of this individual are written into an internal matrix. The collected individuals can also be written into a text file (see Special.CollectBest.WriteFile and Special.CollectBest.FileName).

A value of 0 switches the collection of individuals off. A value of 1 collects individuals every generation. Higher values collect individuals only every defined generation.

- Default value: 0
- Type: integer in [0, Inf]
- Example - collect the best individuals in every 3rd generation:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Interval', 3);
```
- Option number in previous versions: not available (2.x and 1.x)

Special.CollectBest.Rate

This option defines the fraction of the whole population or the number of individuals to be collected as best individuals during a collection. Values smaller than 1 define a fraction of the population size. Values of 1 and larger define directly the number of individuals.

- Default value: 0.1 (10%)
- Type: scalar in [0, 1) or integer in [1 Inf]
- Example - set collection rate to 15% and collect every 2 generations:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Interval', 2
                           'Special.CollectBest.Rate', 0.15);
```
- Example - set collection rate to 4 individuals and collect every generation:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Interval', 1
                           'Special.CollectBest.Rate', 4);
```
- Option number in previous versions: not available (2.x and 1.x)

Special.CollectBest.Compare

This option defines the comparison method for the good individuals. This ensures only distinctive individuals are collected. The individuals of the current generation are compared to each other and to the individuals already collected in previous generations. Two individuals can be compared for identity according to their variables values and/or their objective values (even multiple objective values). If the comparison yields the individuals are identical in all values, then this individual is not added to the collection.

- Default value: 0
- Type: integer in {0, 1, 2}
- Three variants are available:
  - 0: check only for identical variable values
  - 1: check for identical objective values and identical variable values
  - 2: check only for identical objective values
- Example - compare the objective values and the variable values of the individuals before collection:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Compare', 1);
```
- The least restrictive method is comparing variable values and objective values. Then an individual is even collected, when another individual produced the same objective value (objective value is identical, but not the variable values). The other way around, when one individual produces different objective values (disturbed objective function), then the two (identical according to their variable values) can still be added to the collection.
- Example: When different individuals according to their variable values are needed then compare just the variable values:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Compare', 0);
```

- Example: When only one individual per found good objective value is needed, then comparing the objective values is best:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.Compare', 2);
```
- Option number in previous versions: not available (2.x and 1.x)

Special.CollectBest.WriteFile

This option switches writing the collected individuals to a text file on. The name of the file is defined in Special.CollectBest.FileName.

- Default value: 0
- Type: integer in {0, 1}
- Two variants are available:
  - 0: do not write the collected individuals to a text file
  - 1: write the collected individuals to a text file
- Example - switch writing collected individuals to a text file on:
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.WriteFile', 1);
```
- Option number in previous versions: not available (2.x and 1.x)

Special.CollectBest.FileName

This option contains the file name to save the collected best individuals. An optional relative or absolute path may be added (employing Matlab conventions). If no path is defined, the current working directory is used.

- Default value: 'CollectBestInd'
- Type: string
- Example - set the file name for best individuals to 'ColInd.txt' and add the current date and time at the end of the file name (using straddtime):
```
GeaOpt = geaoptset(GeaOpt, 'Special.CollectBest.FileName',
                           straddtime('ColInd.txt'));
% filename: ColInd_2000_Aug_27_14-32.txt
```
- Example of file with collected best individuals (objfun1, 5 variables, collect 4 individuals). The header shows the number of the objective and the variables. Each line contains one individual. The best individual of every collection phase comes first, the next best follow. Here, 4 individuals are added during each collection phase. The first 3 collections are shown:

1. objv	1. var	2. var	3. var	4. var	5. Var
65438	-189.83	97.117	-62.709	-90.832	-88.249
68964	-127.04	-92.959	-135.3	122.15	-104.67
81624	-7.2665	-37.553	80.748	253.08	-97.927
1.0767e+005	124.55	63.665	154.57	6.5249	253.32
21843	-122.04	69.944	-7.5338	-43.448	10.678
43095	-162.7	86.244	-40.631	-71.871	-48.664
54318	-154.47	73.266	-48.199	-106.82	-106.56
73896	-127.64	224.38	-77.343	19.578	-29.873
36771	-145.71	105.68	26.7	59.601	-10.325
50791	-148.92	104.4	-54.164	-80.84	-90.809
52756	-175.42	91.341	-50.982	-80.761	-67.224
60138	-184.06	94.804	-58.013	-86.799	-79.829

- The utility function straddtime is part of the GEATbx. It is useful for extending a static file name with the current date and time when running the same optimization multiple times. The results of each run are separate.
- Option number in previous versions: not available (2.x and 1.x)

13 Comparison of Options (1.9x / 2.x / 3.x)

This Chapter contains an comparison overview of the options and their numbers / names in version 1.9x / 2.x and 3.x. The main aim of this comparison is a simple means to update older optimization setups to version 3.x.

This table contains a textual description of every option in column one. The table is sorted according to the option numbers of version 1.9x (2. column) and 2.x (3. column). The name of the option in version 3.x is contained in the 4. Column and linked to the full description above.

This table is a good overview of the available options in each version of GEATbx. It shows also, if an option was internal (int. - used inside the toolbox, but not contained in the options vector) or not available (n.a.).

Tab. 13-1: Comparison table of options (version 1.9x, 2.x and 3.x)

Description	1.9x	2.x	Name in 3.x
selection function (name)	5	5	`Selection.Name`
mutation function (name)	6	6	`Mutation.Name`
recombination function (name)	7	7	`Recombination.Name`
mutation rate	int.	11	`Mutation.Rate`
recombination rate	int.	12	`Recombination.Rate`
variable format	19	19	`VariableFormat`
number of individuals	20	20	`NumberIndividuals`
number of subpopulations	21	21	`NumberSubpopulation`
generation gap	22	22	`Selection.GenerationGap`
reinsertion rate	int.	23	`Selection.ReinsertionRate`
mutation precision	int.	25	`Mutation.Precision`
selection pressure	23	26	`Selection.Pressure`
ranking method	int.	int.	`Selection.RankingMethod`
ranking multi-objective	n.a.	n.a.	`Selection.RankingMultiobj`
dimension of local selection	n.a.	27	`Selection.LocalDimension`
topology of local selection	n.a.	28	`Selection.LocalTopology`
distance of local selection	n.a.	29	`Selection.LocalDistance`
reinsertion method	int.	30	`Selection.ReinsertionMethod`
do migration	int.	31	`Migration.Do`
migration rate	26	32	`Migration.Rate`
migration interval	int.	33	`Migration.Interval`
migration topology	27	34	`Migration.Topology`
migration selection	int.	int.	`Migration.Selection`
do competition	n.a.	36	`Competition.Do`
competition rate	n.a.	37	`Competition.Rate`
competition interval	n.a.	38	`Competition.Interval`
subpop minimum for competition	n.a.	39	`Competition.SubpopMinimum`
termination method	n.a.	50	`Termination.Method`
maximal number of generations	14	51 (14)	`Termination.MaxGenerations`

Description	1.9x	2.x	Name in 3.x
termination by max. optimization time	n.a.	52	`Termination.MaxTime`
termination by difference to optimum	n.a.	53 (3)	`Termination.Diff2Optimum`
termination by running mean	n.a.	55/54	`Termination.RunningMean`
termination by standard deviation	n.a.	54/55	`Termination.StdObjV`
termination by good/worst obj. value	n.a.	56	`Termination.GoodWorstObjV`
termination by phi	n.a.	57	`Termination.Phi`
termination by kappa	n.a.	58	`Termination.Kappa`
termination by cluster analysis	n.a.	n.a.	`Termination.Cluster`
interval for text output on screen	1	1	`Output.TextInterval`
interval for graphical output	1	61	`Output.GrafikInterval`
methods (plot types) of graphical output	n.a.	62	`Output.GrafikMethod`
styles of graphical output	n.a.	63	`Output.GrafikStyle`
interval for text output in file	28	41	`Output.SaveTextInterval`
file name for saving text output	n.a.	GLOBAL_ FILERESULTS	`Output.SaveTextFileName`
interval for saving binary data	n.a.	44	`Output.SaveBinDataInterval`
file name for saving binary data	n.a.	GLOBAL_ FILEMAT	`Output.SaveBinDataFileName`
interval for specific visualization	30	43	`Output.StatePlotInterval`
function name for specific visualization	n.a.	GLOBAL_ STATEPLOTFUN	`Output.StatePlotFunction`
excludes parts of screen status information	n.a.	int.	`Output.TextExclude*`
excludes part of saved status information	n.a.	int.	`Output.SaveTextExclude*`
best objective value(s) found	8	8	`Run.BestObjectiveValue`
number of objective function evaluations	10	10	`Run.CountObjFun`
number of current generation	int.	int.	`Run.Generation`
termination criteria reached	n.a.	int.	`Run.DoTerminate`
function name of objective function	int. (OBJ_F)	int. (OBJ_F)	`System.ObjFunFilename`
boundaries of (problem) variables	int. (VLB, VUB)	int. (VLB, VUB)	`System.ObjFunVarBounds`
additional parameters for obj. function	int. (P1-P10)	int. (P1-P15)	`System.ObjFunAddPara`
define soft boundaries for variables	n.a.	n.a.	`System.ObjFunVarBoundOut`
goals for multiple objectives	n.a	n.a.	`System.ObjFunGoals`
best objective value (minimum)	int.	int.	`System.ObjFunMinimum`
textual description of objective function	int.	int.	`System.ObjFunDescription`
keep preinitialized individuals	n.a.	n.a.	`Special.InitPresetKeep`
create individuals uniform at random	n.a.	n.a.	`Special.InitUniformCreate`
randomized preinitialized individuals	n.a.	n.a	`Special.InitPresetRand`
do special initialization	29	42	`Special.InitDo` (obsolete)
function name for special initialization	n.a.	GLOBAL_INITFUN	`Special.InitFunction` (obsolete)
interval for collecting best individuals	n.a.	n.a.	`Special.CollectBest.Interval`
rate/number of individuals to collect	n.a.	n.a.	`Special.CollectBest.Rate`
how to compare individuals for identity	n.a.	n.a.	`Special.CollectBest.Compare`
write individuals to file	n.a.	n.a.	`Special.CollectBest.WriteFile`
file name for collected individuals	n.a.	n.a.	`Special.CollectBest.FileName`

Index

T

V